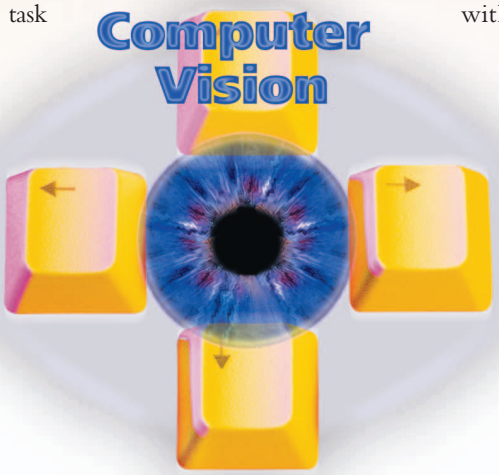
ViSP for visual servoing: a generic software platform with a wide class of robot control skills

E. Marchand, F. Spindler, François Chaumette

ViSP for Visual Servoing

A Generic Software Platform with a Wide Class of Robot Control Skills

Several software packages or toolboxes written in various languages have been proposed in order to simulate robotic manipulator control. The MATLAB Robotics Toolbox [8] allows for the simple manipulation of serial-link manipulator; Roboop [14] is a manipulator simulation package (written in C++). On the other hand, only a few systems allow a simple specification and the execution of a robotic task on a real system. Some systems, though not always related to vision-based robotics, have been described in [9].



IRIS: ©1998 CORBIS CORPORATION, COMPUTER KEYS © DIGITAL VISION

Visual servoing is a very important research area in robotics. Despite all the research in this field, it seems there is no software environment that allows the fast prototyping of visual servoing tasks. The main reason is that it usually requires specific hardware (the robot and specific framegrabbers). Consequently, the resulting applications are not portable and can only adapted to other environments. Today's software design allows us to propose elementary components that can be combined to build portable high-level applications. Furthermore, the increasing speed of microprocessors allows the development of real-time image processing algorithms on a simple workstation. A visual servoing toolbox for MATLAB/Simulink [3] has been recently proposed, but it features only simulation capabilities.

Chaumette et al. [6] proposed a "library of canonical vision-based tasks" for visual servoing that catalogs the most classical linkages. Toyama and Hager describe in [32] what such a system should be in the case of stereo visual servoing. The presented system (called Servomatic) is specified using the same philosophy as the XVision system [16] and would have been independent from the robot and the tracking algorithms. Following these precedents, ViSP (that is, Visual Servoing Platform), the software environment we present in this article, features all these capabilities: independence with respect to the hardware, simplicity, extendibility, and portability. Moreover, ViSP features a large library of elementary tasks with various visual features that can be combined together, an image processing library that allows the tracking of visual cues at video rate, a simulator, an interface with various classical framegrabbers, etc. The platform is implemented in C++ under Linux.

ViSP: Overview and Major Features

Control Issue

Visual servoing techniques consist of using the data provided by one or several cameras in order to control the motion of a robotic system [13], [19]. A large variety of positioning or target tracking tasks can be implemented by controlling from one to all degrees of freedom of the system. Whatever the sensor configuration, which can vary from one camera mounted on the robot end effector to several free-standing cameras, a set of visual features $\mathbf{s}$ must be designed from the visual measurements $\mathbf{x}(t)$ ($\mathbf{s} = \mathbf{s}(\mathbf{x}(t))$), allowing control of the desired degrees of freedom. A control law also must be designed so that these features $\mathbf{s}$ reach a desired value $\mathbf{s}^*$, defining a correct realization of the task. A desired trajectory $\mathbf{s}^*(t)$ can also be tracked [1], [29]. The control principle is thus to regulate the error vector $\mathbf{s} - \mathbf{s}^*$ to zero.

BY ÉRIC MARCHAND, FABIEN SPINDLER, AND FRANÇOIS CHAUMETTE

1070-9932/05/$20.00©2005 IEEE

Control Law

To ensure the convergence of $\mathbf{s}$ to its desired value $\mathbf{s}^*$, we need to model or approximate the interaction matrix $\mathbf{L_s}$, which links the time variation of the selected visual features to the relative camera-object kinematics screw $\mathbf{v}$ and is defined by the classical equation [13].

$$\dot{\mathbf{s}} = \mathbf{L_s}\mathbf{v}. \tag{1}$$

If we want to control the robot using the joint velocities, we have

$$\dot{\mathbf{s}} = \mathbf{J_s}\dot{\mathbf{q}} + \frac{\partial \mathbf{s}}{\partial t}, \tag{2}$$

where $\mathbf{J_s}$ is the features Jacobian and where $\partial\mathbf{s}/\partial t$ represents the variation of $\mathbf{s}$ due to potential motion of the object (for an eye-in-hand system) or to potential motion of the camera (for an eye-to-hand system). More precisely, if we consider an eye-in-hand system we have

$$\mathbf{J_s} = \mathbf{L_s}{}^{c}\mathbf{V}_{n}{}^{n}\mathbf{J}_{n}(\mathbf{q}), \tag{3}$$

where

- ${}^{n}\mathbf{J}_{n}(\mathbf{q})$ is the robot Jacobian expressed in the end-effector frame R_n
- ${}^{c}\mathbf{V}_{n}$ allows the transformation of the velocity screw between the camera frame R_c and the end-effector frame R_n. It is given by

$${}^{c}\mathbf{V}_{n} = \begin{bmatrix} {}^{c}\mathbf{R}_{n} & [{}^{c}\mathbf{t}_{n}]_{\times}{}^{c}\mathbf{R}_{n} \\ \mathbf{0}_3 & {}^{c}\mathbf{R}_{n} \end{bmatrix}, \tag{4}$$

where ${}^{c}\mathbf{R}_{n}$ and ${}^{c}\mathbf{t}_{n}$ are the rotation and translation between frames R_c and R_n and $[\mathbf{t}]_{\times}$ is the skew matrix related to $\mathbf{t}$. This matrix is constant if the camera is rigidly linked to the end effector.

Now, if we now consider an eye-to-hand system we have

$$\mathbf{J_s} = -\mathbf{L_s}{}^{c}\mathbf{V}_{\mathcal{F}}{}^{\mathcal{F}}\mathbf{J}_{n}(\mathbf{q}), \tag{5}$$

where

- ${}^{\mathcal{F}}\mathbf{J}_{n}(\mathbf{q})$ is the robot Jacobian expressed in the robot reference frame $R_{\mathcal{F}}$
- ${}^{c}\mathbf{V}_{\mathcal{F}}$ allows the transformation of the velocity screw between coordinate frames (here the camera frame R_c and the robot reference frame $R_{\mathcal{F}}$). This matrix is constant if the camera is motionless.

In all cases, a control law that minimizes the error $\mathbf{s} - \mathbf{s}^*$ is then given by

$$\dot{\mathbf{q}} = -\lambda\widehat{\mathbf{J}_{\mathbf{s}}^{+}}(\mathbf{s} - \mathbf{s}^*) - \widehat{\frac{\partial \mathbf{s}}{\partial t}}, \tag{6}$$

where λ is the proportional coefficient involved in the exponential convergence of the error and $\widehat{\partial\mathbf{s}/\partial t}$ is an estimation of the object/camera motion. ViSP allows the consideration of both eye-in-hand and eye-to-hand configurations. Finally, note that a secondary task $\mathbf{e_2}$ can be simply added (as reported in "Introduce More Complex Image Processing and a Secondary Task") when all the degrees of freedom are not constrained by the visual task. In that case, we have [13]

$$\dot{\mathbf{q}} = -\lambda\left(\mathbf{W}^{+}\mathbf{W}\widehat{\mathbf{J}_{\mathbf{s}}^{+}}(\mathbf{s} - \mathbf{s}^*) + (\mathbf{I} - \mathbf{W}^{+}\mathbf{W})\mathbf{e_2}\right) + (\mathbf{I} - \mathbf{W}^{+}\mathbf{W})\frac{\partial \mathbf{e_2}}{\partial t}, \tag{7}$$

where $\mathbf{W}^{+}$ and $\mathbf{I} - \mathbf{W}^{+}\mathbf{W}$ are projection operators that guarantee the camera motion due to the secondary task is compatible with the regulation of $\mathbf{s}$ to $\mathbf{s}^*$.

If we consider here, without loss of generality, the case of an eye-in-hand system observing a motionless target we have

$$\dot{\mathbf{s}} = \mathbf{L_s}\mathbf{v}, \tag{8}$$

where $\mathbf{v} = {}^{c}\mathbf{V}_{n}{}^{n}\mathbf{J}_{n}(\mathbf{q})\,\dot{\mathbf{q}}$ is the camera velocity. If the low-level robot controller allows $\mathbf{v}$ to be sent as inputs, a simple control law can be obtained:

$$\mathbf{v} = -\lambda\widehat{\mathbf{L}_{\mathbf{s}}^{+}}(\mathbf{s} - \mathbf{s}^*), \tag{9}$$

where $\widehat{\mathbf{L}_{\mathbf{s}}}$ is a model or an approximation of the interaction matrix. It can be chosen as [4]

- $\widehat{\mathbf{L}_{\mathbf{s}}} = \widehat{\mathbf{L}_{\mathbf{s}}}(\mathbf{s}, \mathbf{r})$, where the interaction matrix is computed at the current position of the visual feature and the current three-dimensional (3-D) pose (denoted $\mathbf{r}$) if it is available
- $\widehat{\mathbf{L}_{\mathbf{s}}} = \widehat{\mathbf{L}_{\mathbf{s}}}(\mathbf{s}^*, \mathbf{r}^*)$, where the interaction matrix is computed only once at the desired position of $\mathbf{s}$ and $\mathbf{r}$
- $\widehat{\mathbf{L}_{\mathbf{s}}} = 1/2(\widehat{\mathbf{L}_{\mathbf{s}}}(\mathbf{s}, \mathbf{r}) + \widehat{\mathbf{L}_{\mathbf{s}}}(\mathbf{s}^*, \mathbf{r}^*))$, as reported in [23].

These possibilities have been integrated in ViSP, but other possibilities (such as a learning process of the interaction matrix [18], [20], [21]) are always possible, although they are not currently integrated in the software. Let us point out that in this case (7) can be rewritten by [13]

$$\mathbf{v} = -\lambda\left(\mathbf{W}^{+}\mathbf{W}\widehat{\mathbf{L}_{\mathbf{s}}^{+}}(\mathbf{s} - \mathbf{s}^*) + (\mathbf{I} - \mathbf{W}^{+}\mathbf{W})\mathbf{e_2}\right) + (\mathbf{I} - \mathbf{W}^{+}\mathbf{W})\frac{\partial \mathbf{e_2}}{\partial t}. \tag{10}$$

A Library of Visual Servoing Skills

With a vision sensor providing two-dimensional (2-D) measurements $\mathbf{x}(t)$, potential visual features $\mathbf{s}$ are numerous. Two-dimensional data (such as coordinates of feature points in the image), as well as 3-D data (provided by a localization algorithm and exploiting the extracted 2-D measurements), can be considered. It is also possible to combine 2-D and 3-D visual features to exploit the advantages of each approach while avoiding their respective drawbacks [24].

A systematic method has been proposed to derive analytically the interaction matrix of a set of visual features defined upon geometric primitives [13], [5], [6]. Any kind of visual features can be considered within the same formalism (coordinates of

points, straight line orientation, area, or more generally, image moments, distance, etc.) Knowing these interaction matrices, the construction of elementary visual servoing tasks is straightforward. As explained in [6] a large library of elementary skills can be proposed. The current version of ViSP has the following predefined visual features:

- 2-D visual features: 2-D points, 2-D straight lines, 2-D ellipses
- 3-D visual features: 3-D points, 3-D straight lines, $\theta\mathbf{u}$ where θ and $\mathbf{u}$ are the angle and the axis of the rotation that the camera must realize. These visual features are useful for 3-D [34] or 2 1/2-D visual servoing [24].

Using these elementary visual features, more complex tasks can be considered by stacking the elementary interaction matrices.

For example, if we want to build a 2 1/2-D visual servoing task defined by $\mathbf{s} = (x, y, \log(Z/Z^*), \theta\mathbf{u})$, where (x, y) are the coordinates of a 2-D point, Z/Z^* is the ratio between the current and the desired depth of the point and desired position, and where θ and $\mathbf{u}$ are the angle and the axis of the rotation that the camera must realize, the resulting interaction matrix is given by

$$\mathbf{L_s} = \begin{bmatrix} \mathbf{L_p} \\ \mathbf{L_z} \\ \mathbf{L}_{\theta\mathbf{u}} \end{bmatrix}, \tag{11}$$

where $\mathbf{L_p}$, $\mathbf{L_z}$, and $\mathbf{L}_\theta\mathbf{u}$ have the following forms:

$$\mathbf{L_p} = \begin{bmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{bmatrix}, \tag{12}$$

$$\mathbf{L_z} = [\,0 \quad 0 \quad -1/Z \quad -y \quad x \quad 0\,], \tag{13}$$

$$\mathbf{L}_{\theta\mathbf{u}} = [\,\mathbf{0}_3 \quad \mathbf{L}_\omega\,]$$

with

$$\mathbf{L}_\omega = \mathbf{I}_3 - \frac{\theta}{2}[\mathbf{u}]_\times + \left(1 - \frac{\text{sinc}\theta}{\text{sinc}^2\frac{\theta}{2}}\right)[\mathbf{u}]_\times^2. \tag{14}$$

All these elementary visual features are available in ViSP or can be simply built. The complete code that allows this task to be built is given in "Building a 2 1/2-D Visual Servoing Task." This way, more feature-based tasks can be simply added to the library.

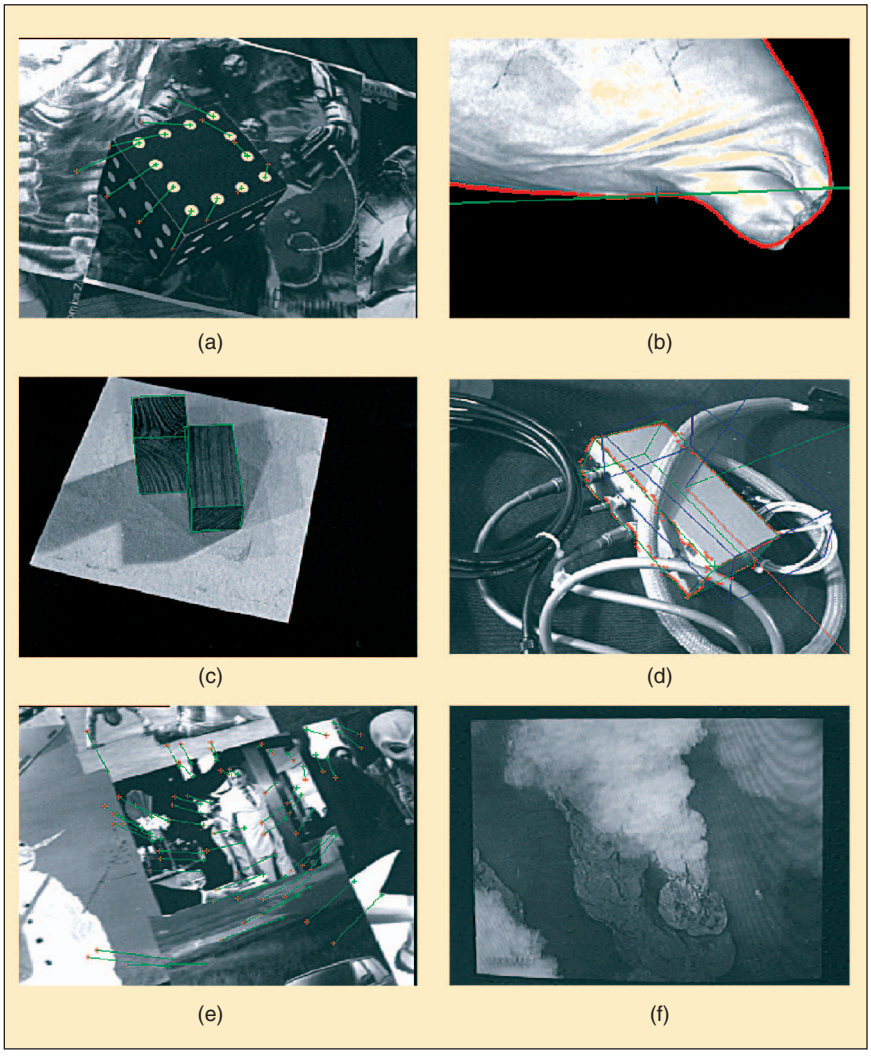


Figure 1. *Increasingly difficult examples of feature tracking in visual servoing experiments.*

Vision-Based Tracking

The definition of objects-tracking algorithms in image sequences is an important issue for research and applications related to visual servoing and more generally for robot vision. A robust extraction and real-time spatio-temporal tracking of visual measurements $\mathbf{x}(t)$ is one of the keys to a successful visual servoing task. To consider visual servoing within large-scale applications, it is now fundamental to consider natural scenes without any fiducial markers and with complex objects in various illumination conditions. From a historical perspective, the use of fiducial markers allowed the validation of theoretical aspects of visual servoing research. If such features are still useful to validate new control laws, it is no longer possible to limit ourselves to such techniques if the final objective is the transfer of these technologies in the industrial world.

Most of the available tracking techniques can be divided into two main classes: feature-based and model-based tracking (see Figure 1). The former approach focuses on tracking 2-D features such as geometric primitives (points, segments, ellipses) or object

contours, regions of interest, etc. The latter explicitly uses a 3-D model of the tracked objects. This second class of methods usually provides a more robust solution (for example, it can cope with partial occlusions of the object). If a 3-D model is available, tracking is closely related to the pose estimation and is then suitable for any visual servoing approach. The main advantage of the 3-D model-based methods is that the knowledge about the scene allows improvement of robustness and performance by predicting hidden movement of the object and reducing the effects of outliers. Another approach may also be considered when the scene is too complex (due to texture or the lack of a specific object). This approach is not based on feature extraction and tracking as in the other two cases but on the analysis of the motion in the image sequence. Two-dimensional motion computation provides interesting information related to both camera motion and scene structure that can be used within a visual servoing process.

Fiducial Markers

Most of articles related to visual servoing consider very basic image-processing algorithms. Indeed, the basic features considered in the control law are usually 2-D points' coordinates. Therefore, the corresponding object is usually composed of "white dots on a black background." Such a choice allows the use of various real-time algorithms (e.g., [33]). The main advantage of this choice is that tracking is very robust and very precise. It is then suitable for all visual servoing control laws (2-D but also 2 1/2-D and 3-D since the position between camera and target can easily be obtained using a pose computation algorithm). From a practical point of view, such algorithms are still useful to validate theoretical aspects of visual servoing research or for educational purposes. Furthermore, in some critical industrial processes, such a simple approach ensures the required robustness (see Figure 2).

2-D Contour Tracking

In order to address the problem of 2-D geometric feature tracking, it is necessary to consider at the low level a generic framework that allows the local tracking of edge points. From the set of tracked edges, it is then possible to perform a robust estimation of features parameters using an iteratively reweighted least-squares method based on robust M-estimation.

For the first point, few systems allow real-time capabilities on a simple workstation. The XVision system is a nice example of such a system [16]. In our case, we decided to use the moving edges (ME) algorithm, which is adapted to the tracking of parametric curves [2]. It is a local approach that allows the matching of moving contours. When dealing with low-level image processing, the contours are sampled at a regular distance. At these sample points, a one-dimensional search is performed to the normal of the contour for corresponding edges. An oriented gradient mask is used to detect the presence of a similar contour. One of the advantages of this method is that it only searches for edges that are oriented in the same direction as the parent contour. This is therefore implemented with convolution efficiency and leads to real-time performance.

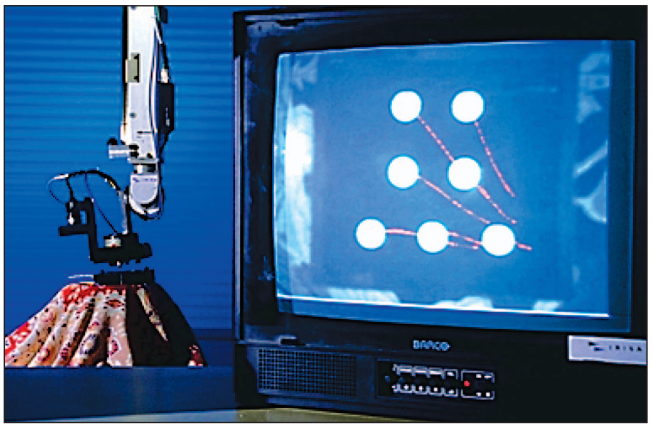


Figure 2. *Visual servoing using fiducial markers for a grasping task. The image was acquired by the camera on the front and the eye-in-hand camera on the back.*

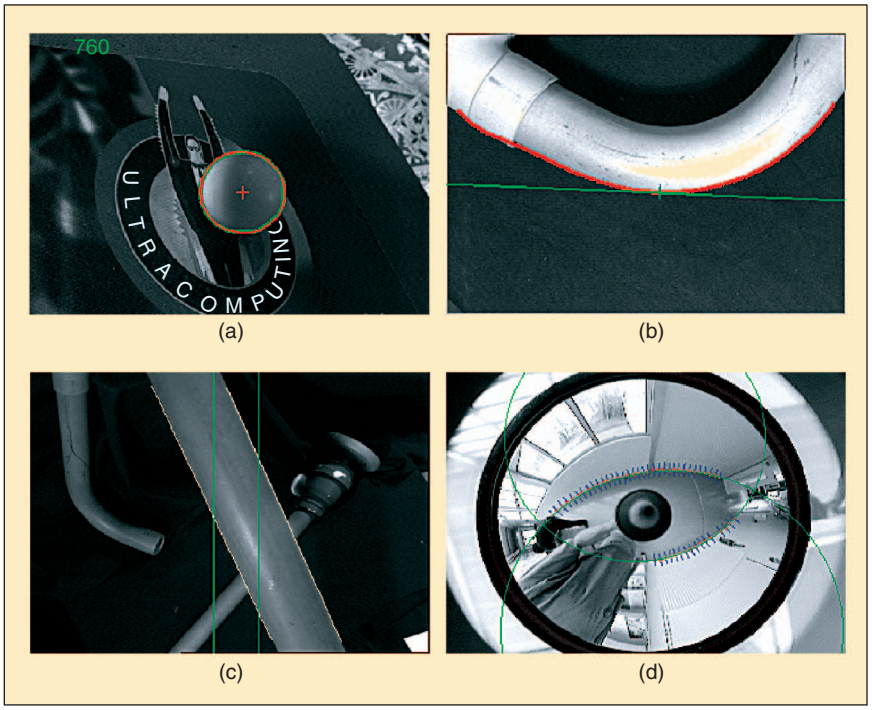


Figure 3. *Tracking 2-D features using the Moving Edges algorithm within visual servoing experiments: (a) the 3-D reconstruction of a sphere using active vision, (b) contour following, (c) positioning with a cylinder with joint limits avoidance, and (d) ellipses tracking (which corresponds to the projection of 3-D straight lines in catadioptric images).*

Figure 3 shows several results of features tracking (line, circle, contours) in visual servoing experiments that use the ViSP library. The proposed tracking approach based on the ME algorithm allows a real-time tracking of geometric features in an image sequence. It is robust with respect to partial occlusions and shadows. However, as a local algorithm, its robustness is limited in complex scenes with a highly textured environment.

Pose Estimation

In particular visual servoing types (most 3-D visual servoing, some 2 1/2-D visual servoing, and the 2-D visual servoing where the depth information must be recomputed at each iteration), the 3-D pose of the camera with respect to the scene is required. This pose estimation process has been widely considered in the computer vision literature. Purely geometric or numerical and iterative approaches may be considered [11]. Linear approaches use a least-squares method to estimate the pose. Full-scale nonlinear optimization techniques (e.g., [22], [7], [12]) consist of minimizing the error between the observation and the forward projection of the model. In this case, minimization is handled using numerical iterative algorithms such as Newton-Raphson or Levenberg-Marquardt. The main advantage of these approaches is their accuracy. In ViSP, various algorithms are available: mainly, the iterative approach proposed by Dementhon [11], which is suitable for applications that consider fiducial markers, and a full-scale nonlinear optimization based on the virtual visual servoing approach [27].

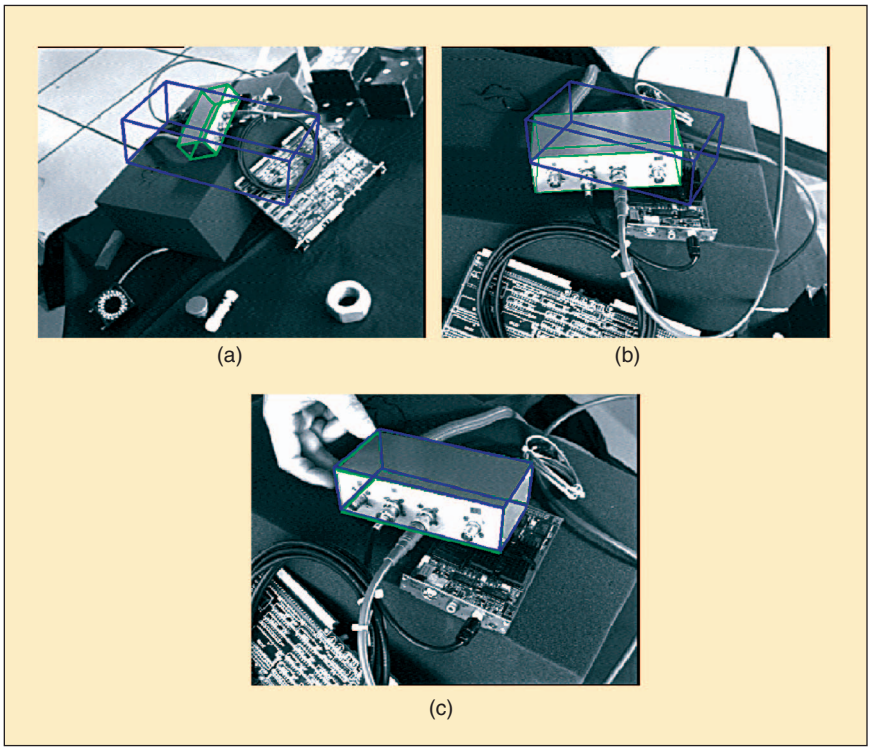


Figure 4. *A 2 1/2-D visual servoing experiment in which the tracked object appears in green and its desired position in blue.*

Other Tracking Capabilities

ViSP also features other tracking capabilities that can be considered within visual servoing experiments [28]. It integrates

- an algorithm to estimate a homography and camera displacement from matched coplanar or not coplanar points [25]
- a version of the Hager Belhumeur [15] tracking algorithm that allows the matching of image templates at video rate.

Though not directly integrated into the software, ViSP provides a direct interface with third-party tracking or image processing algorithms. For example, we propose interfaces

- with a point-of-interests tracker library. We have considered a tracker, built on a differential formulation of a similarity criterion: the well-known Shi-Tomasi-Kanade algorithm [31].
- with a motion estimation algorithm (Motion 2D [30] available in open source), which has been used for motion-based visual servoing [10]. Such image processing algorithms can be used to handle very complex images, as shown on Figure 1(f), and for applications dealing with complex object tracking, the stabilization of a camera mounted on a submarine robot, etc.
- with a 3-D model-based tracking algorithm based on the virtual visual servoing approach [7]. In that case, the image processing is potentially very complex. Indeed, extracting and tracking reliable contour points in real environments is a nontrivial issue. In the experiment presented in Figure 4, images were acquired and processed at video rate (50 Hz). Tracking is always performed at below frame rate (usually in less than 10 ms). All the images given in Figure 4 depict the current position of the tracked object in green while its desired position appears in blue. The considered object is a video multiplexer. It was placed in a highly cluttered environment. Tracking and positioning tasks were correctly achieved. Multiple temporary and partial occlusions were made by hand and various work tools.

Simulation Capabilities

In order to allow fast prototyping of new control laws, ViSP also provides simulation capabilities. Three-dimensional geometric primitives can be forward-projected on the image plane of a virtual camera, and a virtual robot can then be controlled. In order to obtain realistic simulation, it is possible to consider virtual robots with specific Jacobian, joint limits, etc. Furthermore, noise can be added to measures computation and robot motion and in the intrinsic parameters of the camera. An advantage of

this approach with respect to MATLAB simulation (such as the Visual Servoing Toolbox for MATLAB/Simulink [3] or the MATLAB Robotics toolbox [8]) is that the written code can then be used with only minor modifications on a real robot with real images. Figure 5 shows an example of the simulation output. The display interface is written using the Open Inventor library (a C++ layer of OpenGL).

Implementation Issues

As already stated, while developing this software, our goal was to allow a portable (independent from the hardware), fast, and reliable prototyping of visual servoing applications. We also wanted to provide a package that is suitable for real-time implementation and that allows the performance of both simulations and real experiments from the same (or at least very similar) code. Object-oriented programming languages feature these qualities and, therefore, we chose the C++ language for the implementation of ViSP.

The first part of this section presents the internal architecture of the system and how it has been implemented. Describing the full implementation of the software is not in the scope of this article; we will focus on the notion of extendibility and portability. The second part describes how to use the available libraries from an end-user point of view. Let us note that all the functionalities described in this section have been implemented and are fully operational.

An Overview of the ViSP Architecture

To fulfill the extendibility and portability requirements, we divided the platform into three different modules: a module for visual features, visual servoing control laws, and robot controller; a module for image processing, tracking algorithms, and other computer vision algorithms; and a module for a visualization library. Figure 6 summarizes the basic software architecture and module dependencies.

In order to allow fast prototyping of new control laws, ViSP also provides simulation capabilities.

Visual Features and Control Law

Each specific visual feature is derived from a virtual class `vpBaseFeatures`. This class mainly defines a few variables (e.g., a vector that describes the parameters $\mathbf{s}$) and a set of virtual members that are feature dependent (e.g., the way to compute the interaction matrix $\mathbf{L}_\mathbf{s}$ or the virtual function that allows the computation of the visual features $\mathbf{s}$ from the measurements in the image $\mathbf{x}(t)$). It is important to note that all the relations between the control laws library and the visual features library are done through this class. The virtual functions defined in `vpBaseFeatures` can be directly used by the controller `vpServo`, even if they are not yet defined. The consequence is that the controller class never knows the nature of the manipulated features and manipulates only vectors and matrices. Another consequence is that it is not necessary to modify the controller library when adding a new feature. On the other hand, when adding a new feature in

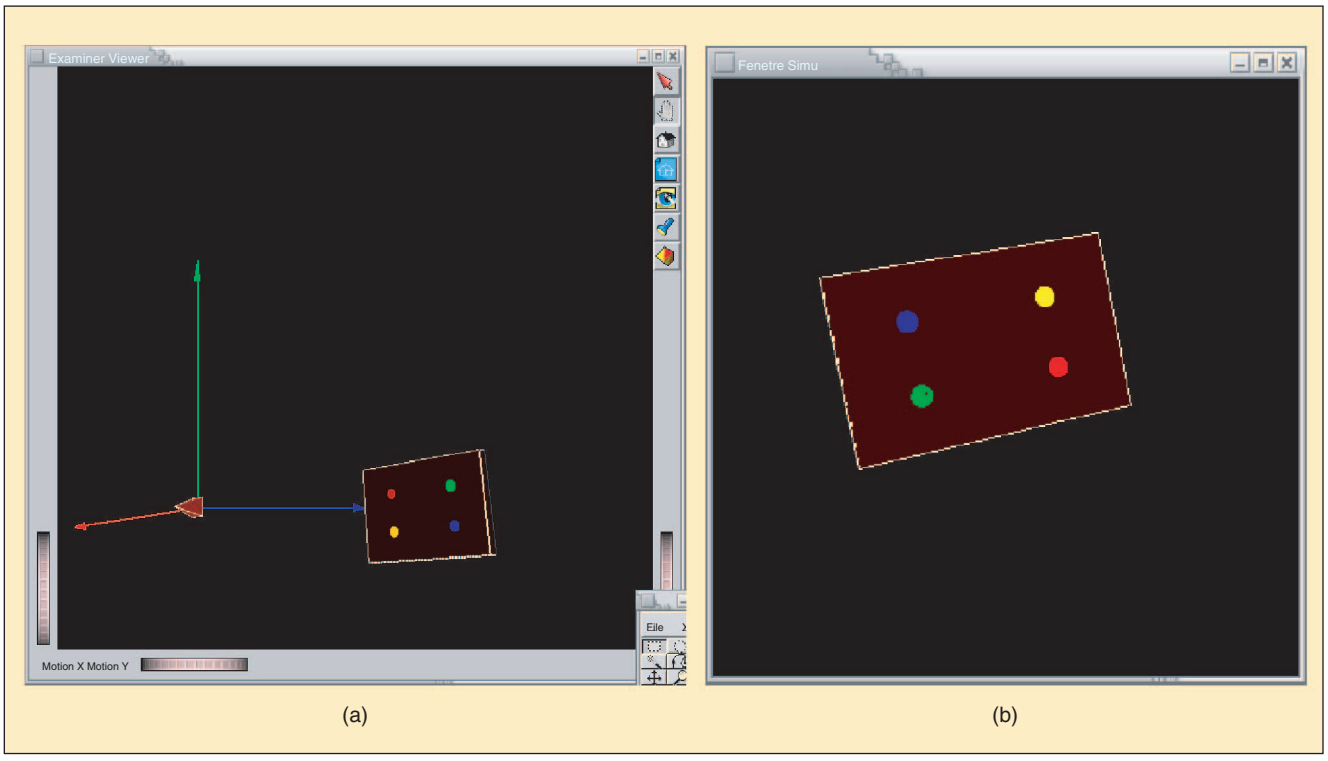


Figure 5. *A ViSP simulation module built using the Open Inventor library: (a) an external view and (b) a view from the camera.*

Today's software design allows us to propose elementary components that can be combined to build portable high-level applications.

the visual features library, the programmer must define the number of its components, the way to compute the interaction matrix, etc. This is done at a lower level (e.g., `vpFeaturePoint` , `vpFeatureLine`.). A generic feature `vpGenericFeature` allows the user to simply define a new feature and to easily test its behavior.

Therefore, the controller (provided in the `vpServo` class) itself provides a generic interface with the visual features. It computes the control law that minimizes the error $\mathbf{s} - \mathbf{s}^*$ as described in "ViSP: Overview and Major Features" according to a list of visual features that defines the task. Various control laws have been implemented as proposed that consider the eye-in-hand or eye-to-hand systems and various formulations of the model of the interaction matrix $\widehat{\mathbf{L}_\mathbf{s}}$. Along with the interaction matrix $\mathbf{L}_\mathbf{s}$, Jacobian $\widehat{\mathbf{J}_\mathbf{s}}$ (which depends on a particular robot, as discussed below) and its pseudoinverse $\widehat{\mathbf{J}_\mathbf{s}^+}$ and its null space, the corresponding projection operator ($\mathbf{W}^+$ and $\mathbf{I} - \mathbf{W}^+\mathbf{W}$) is also computed if a secondary task must be added to the main visual task [see (7)].

Hardware Portability

One of the challenges dealing with a visual servoing package is that it must deal with multiple robotic platforms as well as with various framegrabbers. Obviously, the package does not (and cannot) provide an interface with all possible robots and grabbers, but we built it in order to facilitate adding new hardware.

A new robot class can be derived from the `vpRobot` virtual class. Although `vpRobot` defines the prototypes of each member, it does not provide any interface with a real robot. The new class that must be implemented for each new robot redefines some pure virtual methods defined in `vpRobot`, such as robot motion orders or Jacobian computation generally specific to a given robot and inherits all the methods and attributes of `vpRobot` (i.e., generic control issues). Simulated robots are considered and can be controlled exactly as real robots (specific Jacobian, inverse and forward kinematics, joint limits, and singularities can be modeled and simulated).

Similarly, dealing with framegrabbers, a generic `vpVideo` class has been developed from which a

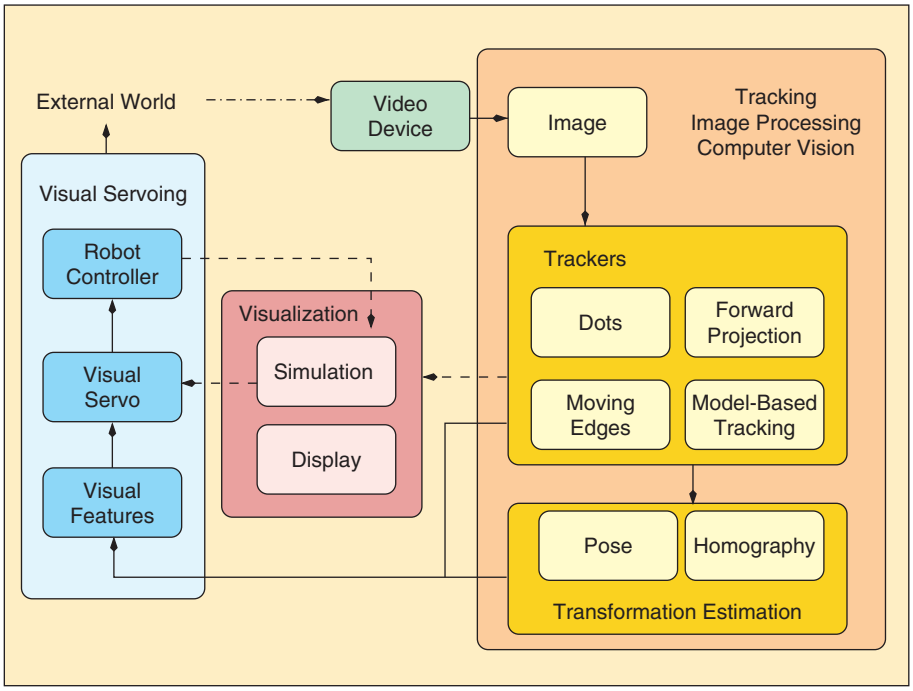


Figure 6. *ViSP software architecture.*

```
1    vpImage<unsigned char> I ;
2
3    vpIeee1394 grabber(VIDEO_PORT) ;  // use the 1394 framegraber
4    grabber.open(I) ;          // associate the grabber to the image
5    grabber.acquire(I) ;
6
7    vpDisplayX display(I,''a new X11 window'') ; // use the X11R6 window system to display the image
8    vpDisplay::display(I) ;                  // associate the display to the image
9
10   vpRobotAfma6 rob ;           // use the Afma6 robot
11   vpCameraParameters cam(u0,v0,px,py) ; // set the value of camera calibration parameters
12
13   vpServo task ;               // create a task
14
15   < code for a specific experiment >
```

Figure 7. *Typical code for ViSP initialization.*

particular framegrabber class can be derived. Some `vpVideo` pure virtual methods must be defined within this new class (mainly initialization, acquisition, closing methods). Such an interface with ViSP is very simple to add since such methods should already exist on the user's system. In the current version of ViSP, some classical framegrabbers are already supported (IEEE 1394, Video4Linux2, Matrox Meteor, IT ICcomp). Along with these acquisition capabilities, ViSP provides various classes to display images using either the X11 system or higher-level libraries such as OpenGL or QT.

ViSP is a fully functional modular architecture that allows fast development of visual servoing applications.

Image Processing and Tracking

A template image class `vpImage` is provided. It has allowed the development of the various trackers described in the previous section. Along with elementary image processing functions, it provides an interface with the images acquisition and display classes. Dealing with the tracking algorithms, a virtual class `vpTracker` is defined and is then derived according to each particular tracking algorithms. An interface with the visual features class is provided.

Matrices

Furthermore, C++ provides capabilities to handle matrices operation using a MATLAB-like syntax. Various numerical analysis algorithms (e.g., SVD and LU decompositions provided by the GSL library) are widely used throughout.

```
16    int nbpoint =4 ;
17    vpDot dot[nbpoint] ;                // tracked objects
18    vpFeaturePoint s[nbpoint], sd[nbpoint] ; // current and desired feature (nbpoint points)
19    double xd[nbpoint], yd[nbpoint], Zd[nbpoint] ;
20
21    < initialize desired visual feature xd[], yd[], and Zd[] >
22
23    for (i=0; i<nbpoint; i++)
24    {
25      dot[i].initTracking(I) ; //  initialize the tracking process
26
27      // build current visual features from the tracked objects
28      vpFeatureBuilder::create(s[i],cam, dot[i])  ;
29
30      // Init here the desired visual features s* along with
31      // the 3D information required to compute the interaction matrix
32      sd[i].buildFrom(xd[i],yd[i],Zd[i]) ;
33
34      task.addFeature(s[i],sd[i]) ;   // add point-to-point image constraint
35                                      // defines the list of visual features, the error vector
36                                      // as well as the interaction matrix
37    }
38    task.setServo(vpServo::EYEINHAND_CAMERA) ;
39    task.setInteractionMatrixType(vpServo::DESIRED, vpServo::PSEUDO_INVERSE) ;
```

Figure 8. *An example of task definition: positioning with four points.*

```
40    task.setLambda(0.2) ;  // set the gain λ
41    while(...) {
42      vpColVector v(6) ;      // velocity vector
43      grabber.acquire(I) ;          // acquire a new image
44      for (i=0 ; i < nbpoint ; i++)
45      {
46         dot[i].track(I) ;      // perform the feature tracking and the pixel/meter conversion
47         vpFeatureBuilder::create(s[i],cam, dot[i]);
48      }
49
50      v =  task.computeControlLaw() ;
51
52      // send the computed velocity (expressed in the camera frame) to the robot controller
53      robot.setVelocity(vpRobot::CAMERA_FRAME, v) ;
54    }
```

Figure 9. *Typical code for the visual servoing closed loop.*

ViSP from an End-User Point of View

Our other claim was that ViSP is simple to use. We will describe the software environment, from the end-user point of view, using three simple examples implemented using ViSP.

Build a Basic 2-D Visual Servoing Task

Figure 7 defines a typical initialization process that can be used in most programs using ViSP. These lines define the framegrabber (here an IEEE 1394 camera), the display system (here X11R6), a robot (the 6-DOF Afma gantry robot of IRISA), and a camera (with given calibration parameters) and finally create a task.

Once these initializations have been achieved, the user is ready to define the tracker of the visual cues and the visual servoing task. In this first example, we chose the classical positioning task with respect to four points, using their x and y coordinates in the image. Dealing with the tracking process, in this example we chose to track fiducial markers (`vpDot`). The features (`vpFeaturePoint`) are created from the tracked markers (`vpDot`) using member functions of the `vpFeatureBuilder` class (in this simple case, it mainly achieves a pixel-to-meter conversion). The desired values of the visual feature $\mathbf{s}^*$ are also defined, and a link between the current value (`s[i]`) of the visual feature in the image and the desired value (`sd[i]`) is then created. To initialize `sd`, the `buildFrom` method (that allows x and y 2-D coordinates to be set along with the desired depth Z used in the interaction matrix computation) is used. Each call to the `addFeature` method creates a 2×6 interaction matrix which is "stacked" to the current one. At the end of this process, an 8×6 matrix and the corresponding error vector are then created. Line 38 specifies that we consider an eye-in-hand configuration with velocity computedi n the camera frame. Furthermore, the interaction matrix will be computed at the desired position $\widehat{\mathbf{L}_\mathbf{s}} = \widehat{\mathbf{L}_\mathbf{s}}(\mathbf{s}^*, \mathbf{r}^*)$ (Line 39) and the control law will be computed using the pseudoinverse of the interaction matrix (other possibilities such as considering the transpose of $\mathbf{L}_\mathbf{s}$ also exist, even if we do not recommend their use at all). See Figure 8.

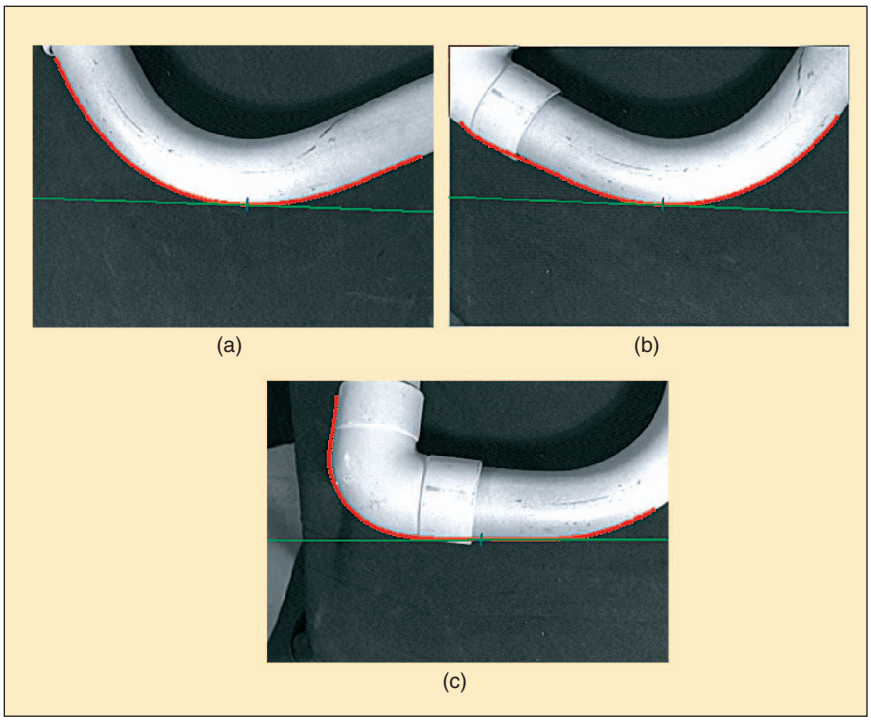


Figure 10. *A pipe-following task: three images acquired during the task with tracked curve (red) and tangent to the curve (green).*

```
1   vpSpline S(CUBICSPLINE) ;  // define a tracker (here track a spline using the  ME algorithm)
2   S.initTracking(I) ;
3
4   vpFeatureLine L ;                  // define the visual feature: a line
5   L = S.tangent(0,0) ;               // defined by the tangent to the spline
6   vpFeatureLine Ld(0,PI/2) ;         // and that must be seen horizontal and centered in the image
7
8   task.addFeature(L,Ld) ;       // define the visual task
9
10  while(...) {
11    vpColVector v(6), g(6)
12    grabber.acquire(I) ;
13    S.track(I) ;              // Track the spline (visual cue)
14    L = S.tangent(0,0) ;      // compute the tangent
15
16    g[0] = -Vx ;               // define secondary task
17    v =  task.computeControlLaw()+ task.addSecondaryTask(g) ;  // v = -λW+WLs+(s - s*) + (I - W+W)gs
18
19    robot.setVelocity(vpRobot::CAMERA_FRAME, v) ;
20  }
```

Figure 11. *Code for a curve-following task: task definition and visual servoing closed loop.*

```
1
2   int nbPoint = 7 ;
3   vpDot dot[nbPoint] ; // current visual feature associated with a tracker
4                        // (here track 7 white dots)
5
6   // init the 3D coordinates (X,Y,Z) of the points in object frame
7   vpPoint point[nbPoint] ;
8   point[0].setWorldCoordinates(-L,-L,0) ;  point[1].setWorldCoordinates(L,-L,0) ;
9   point[2].setWorldCoordinates(L,L,0) ;    point[3].setWorldCoordinates(-L,L,0) ;
10  point[4].setWorldCoordinates(2*L,3*L,0); point[5].setWorldCoordinates(0,3*L,0) ;
11  point[6].setWorldCoordinates(-2*L,3*L,0) ;
12
13
14  vpPose pose ;
15  for (i=0; i<nbPoint; i++)
16  {
17     dot[i].initTracking(I, cam) ;   // initialize the tracking process
18     dot[i].track(I) ;               // get the 2D position of the point in meter
19     vpPixelMeterConversion::convertPoint(cam,  dot[i], point[i])  ;
20     // at this point the 2D coordinates (x,y) and 3D coordinates (X,Y,Z)
21     // are available
22     pose.addPoint(point[i]) ;         // consider this point in the pose computation algorithm
23  }
24
25  vpHomogeneousMatrix cMo ;
26  pose.computePose(vpPose::DEMENTHON, cMo) ;
27  pose.computePose(vpPose::NON_LINEAR, cMo) ;
28
29  cout << ''Pose'' << cMo << endl ;
```

Figure 12. *An example of pose estimation.*

It is then a straightforward task to write the control loop itself. It features the image acquisition (Line 43) and the current visual features computation from the result of the dots tracking (Lines 46 and 47). The task is then automatically updated. Finally the control law given in (9) is computed and the result is sent to the robot controller (see Figure 9).

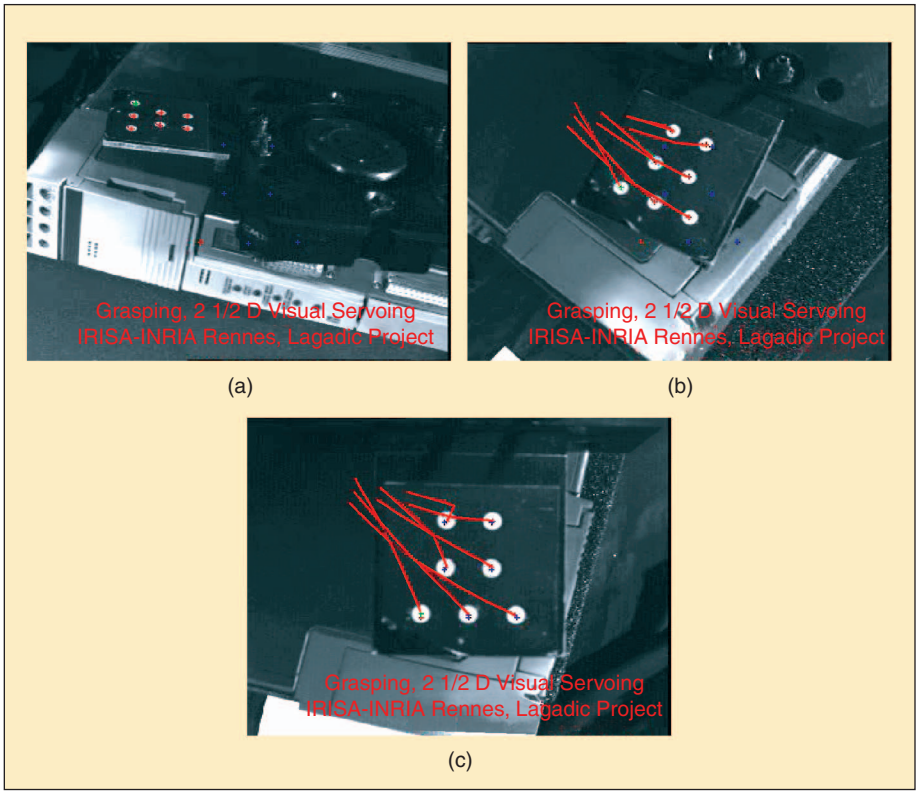


Figure 13. *2 1/2-D visual servoing task as implemented in "Building a 2 1/2-D Visual Servoing Task."*

Introduce More Complex Image Processing and a Secondary Task

Let us consider now a curve-following task [26]. This problem can be divided into two subtasks. The primary task consists of servoing the tangent to the curve (for instance, maintaining this tangent horizontal and centered in the image). The positioning skill used in this experiment is therefore a 2-D line-to-2-D line link and visual features used here are $\mathbf{s} = (\rho, \theta)$, where ρ and θ are the cylindric parameters of the straight line that is the tangent to the curve. A secondary task can be added, and it has been specified as a trajectory tracking at a given constant velocity V_x in the X direction of the camera frame (see Figure 10).

Image processing here consists of tracking a spline in the image sequence and in computing the equation of the tangent to the curve from which we can control the camera

motion. In our software environment, there is no direct relation between the tangent to a curve (that is, here, the measurement $\mathbf{x}$) and a straight line (that is the visual feature $\mathbf{s}$). As explained in the previous example, ViSP normally allows the user to avoid explicit access to the tracker, but the number of relationships among the visual cues and control features is virtually infinite. Therefore, direct access to the trackers is necessary. In this example, the spline tracker is defined in Line 1, whereas the visual features (represented by parameters ρ and θ of a straight line) are associated to this tracker in Line 5. Then in the control loop, the spline is tracked in each frame, and the new visual features are computed (Line 14) and introduced in the task. The secondary task is then considered in Line 17, where $g = -V_x$ corresponds to $\partial \mathbf{e_2}/\partial t$ in (7). For simplicity we have considered that $\mathbf{e_2} = X - X_0 - V_x t$ is always equal to 0. This visual servoing task also features the use of a secondary task. Vector $\mathbf{g}$ is combined with the primary task using the projection operator $\mathbf{I} - \mathbf{W}^+\mathbf{W}$.

This example allows us to show the importance of the three libraries (the trackers library, the visual features library, and the controller library) and how they interact with each other (see Figure 11).

```
55
56      // define the feature ------------------------------------------------
57      vpFeaturePoint p, pd) ;             // 2D reference point
58      vpGenericFeature logZ(1) ;          // log (Z/Z*)
59      vpFeatureThetaU tu ;                // ThetaU
60
61      // build the task (stack the features) -----------------------------
62      task.addFeature(p,pd) ;
63      task.addFeature(logZ) ;
64      task.addFeature(tu) ;    // s = (x, y, log Z/Z*, θu)^T
65
66      //-------------------------------------------------------------------
67      // interaction matrix computed at the current position
68      task.setInteractionMatrixType(vpServo::CURRENT) ;
69
70      // compute Z* and p*
71      vpColVector cP = cdMo*point[0] ;
72      double Zd ;   Zd = cP[2] ;
73      pd.set_x(cP[0]/Zd) ;   pd.set_y(cP[1]/Zd) ;
74
75      while(1) { //-----------------------------------------------------------
76        g.acquire(I) ;
77        vpDisplay::display(I) ;
78
79        // compute the pose using  a non linear minimisation method
80        pose.clearPoint() ;
81        for (i=0 ; i < nbPoint ; i++) {
82          dot[i].track(I) ;
83          vpPixelMeterConversion::convertPoint(cam, dot[i], point[i])  ;
84          pose.addPoint(point[i]) ;
85        }
86        pose.computePose(vpPose::NON_LINEAR, cMo) ;
87
88        // compute the current Z
89        cP = cMo * point[0] ;   Z = cP[2] ;
90        p.buildFrom(point[0].get_x(), point[0].get_y(), Z) ;
91
92        // compute log (Z/Z*) and the corresponding interaction matrix
93        logZ.set_s(log(Z/Zd)) ;
94        vpMatrix LlogZ(1,6) ;
95        LlogZ[0][0] = LlogZ[0][1] = LlogZ[0][5] = 0 ;
96        LlogZ[0][2] = -1/Z ;  LlogZ[0][3] = -p.get_y() ;  LlogZ[0][4] =  p.get_x() ;
97        logZ.setInteractionMatrix(LlogZ) ;
98
99        cdMc = cdMo*cMo.inverse() ; // Compute the displacement to achieve
100       tu.buildFrom(cdMc) ;
101
102       v = task.computeControlLaw() ;
103       robot.setVelocity(vpRobot::CAMERA_FRAME, v) ;
104     }
```

Figure 14. *An example of a 2 1/2-D visual servoing task.*

Building a 2 1/2-D Visual Servoing Task

We now consider the 2 1/2-D visual servoing task presented in "Control Issue." The visual features $\mathbf{s}$ are defined by $\mathbf{s} = (x, y, \log Z/Z*, \theta\mathbf{u})$. To achieve this task, one solution is to consider an estimation of the 3-D position of the point and the camera pose with respect to the object. Figure 12 shows how to get the pose ${}^{c}\mathbf{M}_{o}$ from a set of seven points (also see Figure 13). The class `vpPose` provides functions that compute the pose from a list of points (the list was built using the `addPoint` method). Different methods can be considered in order to compute the pose. In this example, it is first initialized using the Dementhon-Davis [11] approach and improved using a nonlinear minimization method.

Figure 14 shows how to build the 2 1/2-D visual servoing control law. In the first time, we initialized the current and desired value of the visual features. The basic features related to a point and to $\theta\mathbf{u}$ are available in the visual feature library (`vpFeaturePoint` and `vpFeatureThetaU`). However, there is no predefined basic feature for $\log(Z/Z^*)$. In such a case, it is possible to use a generic `vpGenericFeature` feature. The user must then compute, at each iteration, the state vector, the interaction matrix, and the error vector. The task is then built by "stacking" the different visual features using the `addFeature` method. Let us note that when dealing with `logZ` and `tu`, these desired values are zero; therefore, we do not have to specify them (this is implicitly done). Line 68 specifies that the interaction matrix will be computed at the current position $\widehat{\mathbf{L}}_{\mathbf{s}} = \widehat{\mathbf{L}}_{\mathbf{s}}(\mathbf{s}, \mathbf{r})$. In the closed loop itself, we find the point tracking that provides the measurement necessary to compute the pose and the position of the 2-D point (Line 83). The pose is updated from these measurements using a nonlinear minimization method and the displacement $\theta\mathbf{u}$ along with the depth Z of the point are updated (let us note that operator $*$ has been overloaded to allow a simple frame transformation: ${}^{c}\mathbf{P} = {}^{c}\mathbf{M}_{o}\,{}^{o}\mathbf{P}$). The interaction matrix related to $\log Z/Z^*$ must be computed according to (13) (see Line 97). The global task interaction is then updated and the control law computed.

In this example, to compute the depth of the reference point and the rotation that the camera must achieve, we considered a pose estimation process. Let us note that this can also be achieved by the estimation of the homography between the current and the desired image (as explained in [24]). ViSP also features the ability to estimate such homography using various algorithms [17], [24] and to extract from this homography the camera displacement and some useful values such as Z/Z^*.

Conclusions

ViSP is a fully functional modular architecture that allows fast development of visual servoing applications. The platform takes the form of a library which can be divided in three main modules: two are dedicated to control issues (one for control processes and one for canonical vision-based tasks that contains the most classical linkages) and one dedicated to real-time tracking. Let us finally note that ViSP also features a virtual 6-DOF robot that allows the simulation of visual servoing experiments.

ViSP is developed within the INRIA Lagadic project and its kernel, available under the Linux system, is distributed using an open-source license (QPL License). Other modules such as the 3-D model-based tracker are subject to other licenses. The ViSP Web site is located at http://www.irisa.fr/lagadic/visp. The examples given in the article and many others can be found in the software distribution.

Acknowledgments

The authors acknowledge the contribution of the members of the Lagadic group: Nicolas Mansard, Anthony Remazeilles, Andrew Comport, and Muriel Pressigout as well as the former Ph.D. students (including Ezio Malis and Youcef Mezouar) who contributed to the creation of this software.

Keywords

Visual servoing, tracking, C++ library, fast prototyping, simulation.

References

[1] F. Berry, P. Martinet, and J. Gallice, "Turning around an unknown object using visual servoing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., IROS'2000*, Takamatsu, Japan, vol. 1, pp. 257–262.

[2] P. Bouthemy, "A maximum likelihood framework for determining moving edges," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, no. 5, pp. 499–511, May 1989.

[3] E. Cervera, "Visual servoing toolbox for MATLAB/Simulink," 2003 [Online]. Available: http://vstoolbox.sourceforge.net/

[4] F. Chaumette, "Potential problems of stability and convergence in image-based and position-based visual servoing," in *The Confluence of Vision and Control,* no. 237, (*Lecture Notes in Control and Information Sciences)*, D.J. Kriegman, G. Hager, and A.S. Morse, Eds. New York: Springer, 1997, pp. 67–78.

[5] F. Chaumette, "Image moments: A general and useful set of features for visual servoing," *IEEE Trans. Robot.*, vol. 20, no. 4, pp. 713–723, Aug. 2004.

[6] F. Chaumette, P. Rives, and B. Espiau, "Classification and realization of the different vision-based tasks," in *Visual Servoing*, vol. 7, *World Scientific Series in Robotics and Automated Systems*, K. Hashimoto, Ed. Singapore: 1993, pp. 199–228

[7] A.I. Comport, E. Marchand, and F. Chaumette, "Robust model-based tracking for robot vision," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., IROS'04,* Sendai, Japan, vol. 1, pp. 692–697.

[8] P. Corke, "A robotics toolbox for matlab," *IEEE Robot. Automat. Mag.*, vol. 3, no. 1, pp. 24–32, Sept. 1996.

[9] E. Coste-Manière and B. Espiau, "Special issue on integrated architecture for robot control and programming," *Int. J. Robot. Res.*, vol. 17, no. 4, Apr. 1998.

[10] A. Crétual and F. Chaumette, "Visual servoing based on image motion," *Int. J. Robot. Res.*, vol. 20, no. 11, pp. 857–877, Nov. 2001.

[11] D. Dementhon and L. Davis, "Model-based object pose in 25 lines of codes," *Int. J. Comput. Vision*, vol. 15, no. 1–2, pp. 123–141, 1995.

[12] T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, no. 7, pp. 932–946, July 2002.

[13] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *IEEE Trans. Robot. Automat.*, vol. 8, no. 3, pp. 313–326, June 1992.

[14] R. Gourdeau, "Object-oriented programming for robotic manipulator simulation," *IEEE Robot. Automat. Mag.*, vol. 4, no. 3, pp. 21–29, Sep. 1997.

[15] G. Hager and P. Belhumeur, "Efficient region tracking with parametric models of geometry and illumination," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, no. 10, pp. 1025–1039, Oct. 1998.

[16] G. Hager and K. Toyama, "The XVision system: A general-purpose substrate for portable real-time vision applications" *Comput. Vision Image Understanding*, vol. 69 no. 1, pp. 23–37, Jan. 1998.

[17] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, UK: Cambridge University Press, 2001.

[18] K. Hosoda and M. Asada, "Versatile visual servoing without knowledge of true jacobian," in *Proc. IEEE/RSJ Int. Conf on Intell. Robots Syst., IROS'94*, Munich, Germany, pp. 186–193.

[19] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Trans. Robot. Automat.*, vol. 12, no. 5, pp. 651–670, Oct. 1996.

[20] M. Jägersand, O. Fuentes, and R. Nelson, "Experimental evaluation of uncalibrated visual servoing," in *Proc. IEEE Int. Conf. Robot. Automat. ICRA'97*, Albuquerque, NM, vol. 3, pp. 2874–2880.

[21] J.-T. Lapresté, F. Jurie, M. Dhome, and F. Chaumette, "An efficient method to compute the inverse jacobian matrix in visual servoing," in *Proc. IEEE Int. Conf. Robot. Automat., ICRA'04*, New Orleans, LA.

[22] D.G. Lowe, "Fitting parameterized three-dimensional models to images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, no. 5, pp. 441–450, May 1991.

[23] E. Malis, "Improving vision-based control using efficient second-order minimization techniques," in *Proc. IEEE Int. Conf. Robot. Automat., ICRA'04*, New Orleans, LA, vol. 2, pp. 1843–1848.

[24] E. Malis, F. Chaumette, and S. Boudet, "2 1/2 D visual servoing," *IEEE Trans. Robot. Automat.*, vol. 15, no. 2, pp. 238–250, Apr. 1999.

[25] E. Malis, F. Chaumette, and S. Boudet, "2 1/2 D visual servoing with respect to unknown objects through a new estimation scheme of camera displacement," *Int. J. Comput. Vision*, vol. 37, no. 1, pp. 79–97, June 2000.

[26] E. Marchand, "ViSP: A software environment for eye-in-hand visual servoing," in *Proc. IEEE Int. Conf. Robot. Automat., ICRA'99*, Detroit, MI, vol. 4, pp. 3224–3229.

[27] E. Marchand and F. Chaumette, "Virtual visual servoing: A framework for real-time augmented reality," in *EUROGRAPHICS'02 Conf. Proceeding of Computer Graphics Forum*, Saarebrücken, Germany, 2002, vol. 21, no. 3, pp. 289–298.

[28] E. Marchand and F. Chaumette, "Feature tracking for visual servoing purposes," *Robot. Auton. Syst.*, vol. 52, no. 1, pp. 53–70, Jun. 2005.

[29] Y. Mezouar and F. Chaumette, "Path planning for robust image-based control," *IEEE Trans. Robot. Automat.*, vol. 18, no. 4, pp. 534–549, Aug. 2002.

[30] J.-M. Odobez and P. Bouthemy, "Robust multiresolution estimation of parametric motion models," *J. Visual Commun. Image Representation*, vol. 6, no. 4, pp. 348–365, Dec. 1995.

[31] J. Shi and C. Tomasi, "Good features to track," in *Proc. IEEE Int. Conf. Comput. Vision Pattern Recognition, CVPR'94*, Seattle, Washington, pp. 593–600.

[32] K. Toyama, G. Hager, and J. Wang, "Servomatic: A modular system for robust positioning using stereo visual servoing," in *Proc. Int. Conf. Robot. Automat.*, Minneapolis, MN, 1996, pp. 2636–2643.

[33] M. Vincze and C. Weiman, "On optimizing tracking performance for visual servoing," in *Proc. IEEE Int. Conf. Robot. Automat., ICRA'97*, Albuquerque, NM, 1997, vol. 3, pp. 2856–2861.

[34] W. Wilson, C. Hulls, and G. Bell, "Relative end-effector control using cartesian position-based visual servoing," *IEEE Trans. Robot. Automat.*, vol. 12, no. 5, pp. 684–696, Oct. 1996.

Éric Marchand received the Ph.D. degree and the "Habilitation à Diriger des Recherches" in computer science from the University of Rennes 1 in 1996 and 2004, respectively. He spent one year as a postdoctoral associate in the Artificial Intelligence Lab of the Department of Computer Science at Yale University, New Haven, Connecticut. Since 1997, he has been an INRIA research scientist ("Chargé de recherche") at IRISA-INRIA Rennes in the Lagadic project. His research interests include robotics, perception strategies, visual servoing and real-time object tracking. He is also interested in the software engineering aspects of robot programming. More recently, he has been studying new application fields for visual servoing, such as augmented reality and computer animation.

Fabien Spindler graduated from the Engineer National School (ENI, Brest, France) and received the Specialised Mastère's in electronics and aerospace telecom from Sup'Aéro in Toulouse, France, in 1993. Since November 1994, he has been working for INRIA as a research engineer. He is in charge of the material and software management of several robotic vision cells. He also deals with the development and industrialization of image-based software for robotic or video indexing applications and is in charge of the transfer of some of these to industrial or academic partners.

François Chaumette graduated from the École Nationale Supérieure de Mécanique, Nantes, France, in 1987. He received the Ph.D. degree in computer science from the University of Rennes in 1990. Since 1990, he has been with IRISA/INRIA where he is now senior research scientist and head of the Lagadic group. His research interests include robotics and computer vision, especially visual servoing and active perception. Chaumette received the AFCET/CNRS Prize for the best French thesis in automatic control in 1991. He also received with Ezio Malis the 2002 King-Sun Fu Memorial Best IEEE *Transactions on Robotics and Automation* Paper Award. He is currently associate editor of *IEEE Transactions on Robotics*.

Address for Correspondence: Éric Marchand, IRISA-INRIA Rennes, Campus de Beaulieu, 35042 Rennes Cedex, France. E-mail: Eric.Marchand@irisa.fr.